INFRARED VIDEO TRACKING OF UAVS: GUIDED LANDING IN THE ABSENCE OF GPS SIGNALS

By

Logan W. Graves, B.S.

A Project Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electrical Engineering

University of Alaska Fairbanks

May 2019

APPROVED:

Michael C. Hatfield, Committee Chair
Orion S. Lawlor, Committee Member
Dejan Raskovic, Committee Member
Charlie Mayer, Chair
Department of Electrical and Computer Engineering

1. Abstract

Unmanned Aerial Vehicles (UAVs) use Global Positioning System (GPS) signals to determine their position for automated flight. The GPS signals require an unobstructed view of the sky in order to obtain position information. When inside without a clear view of the sky, such as in a building or mine, other methods are necessary to obtain the relative position of the UAV. For obstacle avoidance a LIDAR/SONAR system is sufficient to ensure automated flight, but for precision landing the LIDAR/SONAR system is insufficient for effectively identifying the location of the landing platform and providing flight control inputs to guide the UAV to the landing platform. This project was developed in order to solve this problem by creating a guidance system utilizing an infrared (IR) camera to track an IR LED and blue LEDs mounted on the UAV from a RaspberryPI 3 Model B+. The RaspberryPI, using OpenCV libraries, can effectively track the position of the LED lights mounted on the UAV, determine rotational and lateral corrections based on this tracking, and, using Dronekit-Python libraries, command the UAV to position itself and land on the platform of the Husky UGV (Unmanned Ground Vehicle).

Table of Contents

List of Figures

2. System Pictures

Figure 1 - Husky with UAV Side View

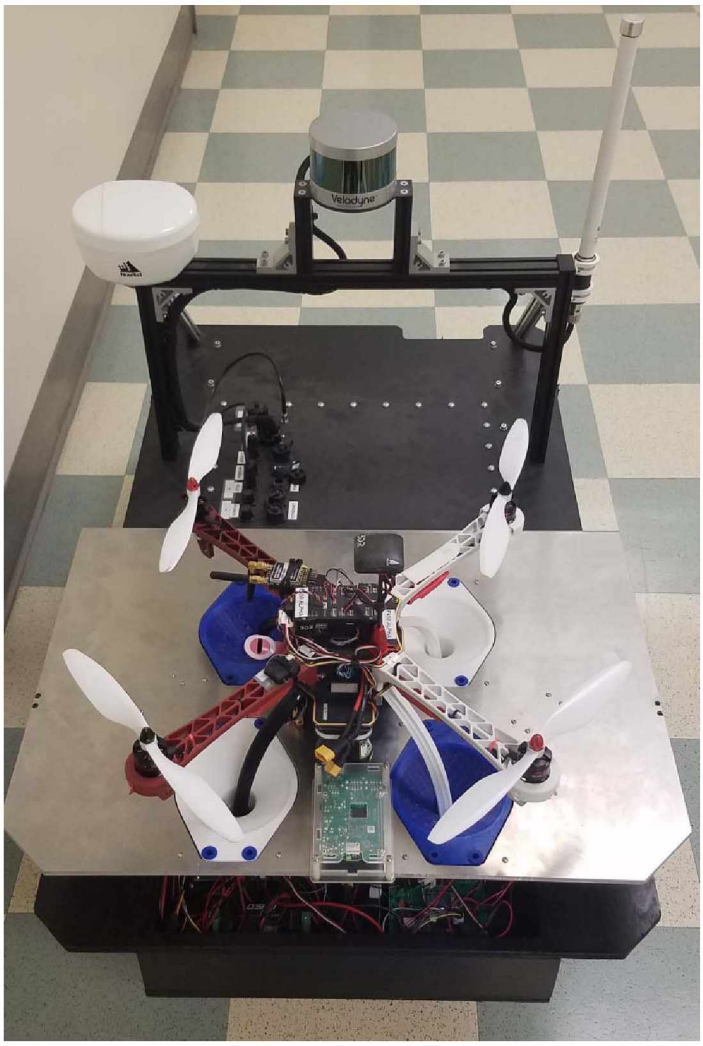

Figure 2 - Husky with UAV Top View

Figure 3 - Bottom View of UAV with Tracking LEDS

3. System Overview

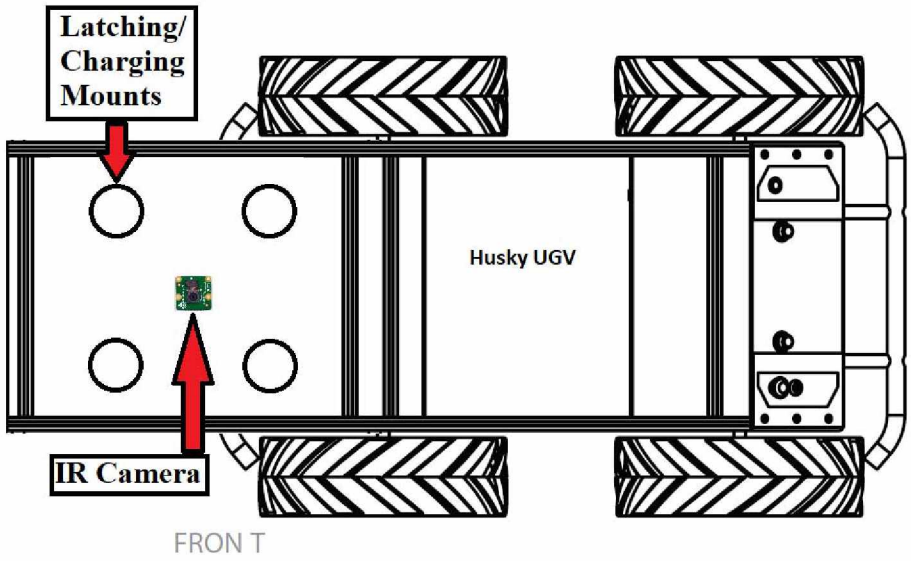

Figure 4 - UGV Functional Diagram

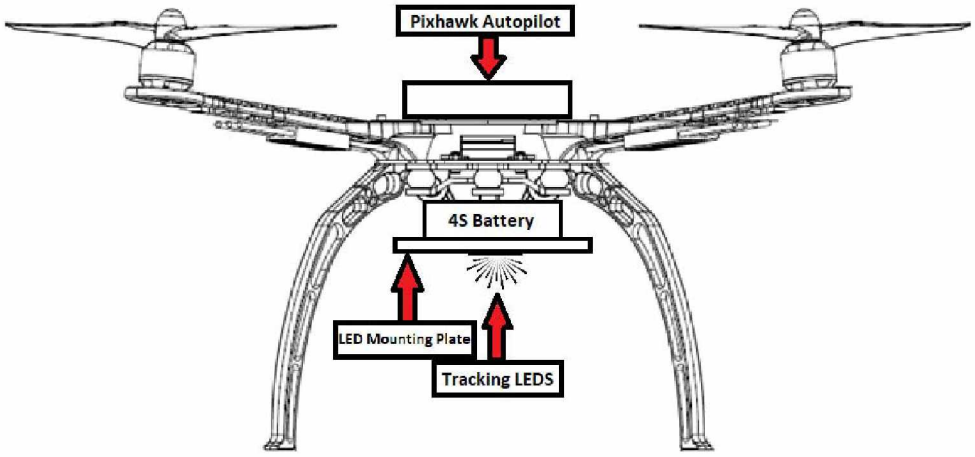

Figure 5 - F450 Functional Diagram

The system itself consists of a Clearpath Husky Unmanned Ground Vehicle (UGV) that has been modified to include a landing/charging platform for a UAV that has been adapted to interface with the charging platform. There are 4 conical receptacles in the landing platform which are used to guide the landing legs of the UAV onto the connections that charge the onboard batteries of the UAV after flight operations. The onboard batteries for the UAV are then charged from supplemental batteries carried by the UGV, allowing for multiple flight

operations in a shorter time period than would be necessary if the UAV had to return to a central location to recharge after flights.

The RaspberryPI is mounted to the landing platform, with the IR camera being mounted centrally between the 4 landing cones. The UGV supplies USB power to the RaspberryPI without the need for modification. The RaspberryPI is connected to an RFD900+, a 1 W, 915 MHz, serial modem, which transmits the flight control commands to the UAV while in flight. The program running on the RaspberryPI is able to send between 5-7 command updates per second to the UAV. The command update rate is limited by how fast the RaspberryPI can capture the image, mask the image, then calculate the position of the LEDs on the (x,y) plane of the image frame.

The UAV is a quad rotor UAV based on the Pixhawk Autopilot system using a DJI 450 ARF frame. It is powered by a 4S, 5200 mAh Li-Po battery. The UAV has been modified with a plastic plate attached underneath on which the tracking LEDs are mounted. The plate extends 15 cm off the front of the aircraft, with the IR LED mounted in the center and the blue LEDs mounted 10 cm further towards the fore of the aircraft. A RFD900+ serial modem is attached to the Pixhawk to receive flight control commands from the RaspberryPI. This radio is in addition to the standard telemetry radio on the Pixhawk, used for UAV to Ground Control Station (GCS) communication. The same radio could be used for both GCS communication and control commands from the RaspberryPI, but this would require sending the RaspberryPI commands through the GCS computer, as well as the responses to queries of UAV flight mode and altitude. This method was not used because of the increased lag and unnecessary complexity this would introduce to the system.

4. System Components

4.1. Raspberry PI 3 Model B+

Figure 6 - RaspberryPI Model3 B+

The RaspberryPI Model3 B+ is a microcomputer based on the Cortex-A53 (ARMv8-A) 64-bit instruction set [1]. It features the Broadcom BCM2837BO quad-core processor at 1.4 GHz. It is equipped with 1GB of LPDDR2 SDRAM, a 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, a Camera Serial Interface (CSI) port for the RaspberryPI NoIR camera, and a MicroSD port for storage. It is running a modified version of Debian called Raspbian, which has been optimized to run on the RaspberryPI. This operating system also provides functionality specific to the RaspberryPI, in the case of this project support for the RaspberryPI NoIR Camera interface. All programming for this project was done in Python v2.7.13.

The RaspberryPI Model3 B+ was chosen for this project for multiple reasons. It is a very small device, which makes it easy to mount and install in the limited space available in the UGV. It has low-power consumption, with max power consumption being around 5 W, which is negligible when being powered from the 480 Wh batteries on the UGV. It has a dedicated interface for the RaspberryPI NoIR Camera, allowing much faster image acquisition than when using a camera through a separate interface, such as USB. The RaspberryPI is also very cheap, with a MSRP of $35, making it very desirable for its capabilities when compared to the price for similar functionality from a laptop or other computer system.

4.2. RaspberryPI NoIR Camera V2

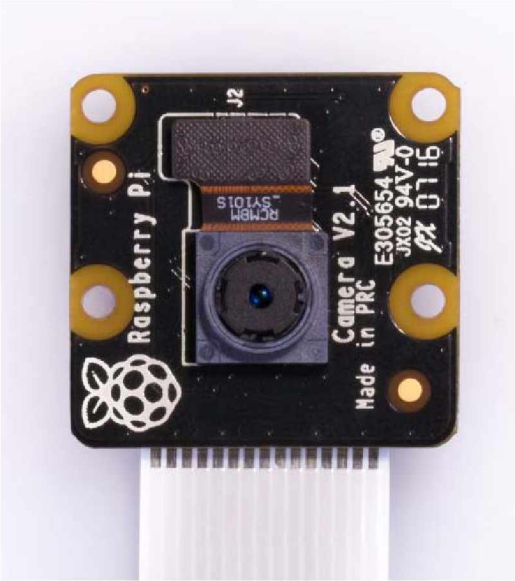

Figure 7 - RaspberryPI NoIR Camera V2

The RaspberryPI NoIR Camera V2 uses a Sony IMX219 8-megapixel sensor and does not employ a physical infrared filter like the majority of cameras [2]. It has a horizontal field of view of 62.2° and a vertical field of view of 48.8°. It was chosen for this project because of its dedicated interface with the RaspberryPI, the availability of the Picamera Python libraries, the V4L2 drivers which allow for unencoded image capture, and the ease of interfacing with OpenCV libraries which will be used for LED tracking.

4.3. Pixhawk v1.0 Autopilot

Figure 8 - Pixhawk v1.0

The autopilot controlling the UAV is the Pixhawk v1.0 running on the ARM Cortex M4 at 180 MHz [3]. This autopilot system has been proven to be very reliable and is easy to interface with due to its open source development. Unlike other commercial autopilot systems, such as DJIs Wukong-M system, its design supports easy modification and provides several communication interfaces, as well as extensive documentation on how to use and modify the operation of those interfaces. The available UART communication interface is used to inject flight control commands from the RaspberryPI, as well as provide feedback to the RaspberryPI on flight mode status.

4.4. Tracking LEDs

Figure 9 - 1W 850 nm InfraRed LED

The central tracking LED on the UAV was chosen to be a 1 W 850 nm InfraRed LED [4]. This LED has a radiant power of 315 mW and a viewing angle of 140°. The LED package, including the pictured heat sink takes up a total area of 14.5 mm x 7.5 mm, making it small enough to easily mount on the UAV. It has a forward voltage of 1.9 V at an operating current of 700 mA, making it easy to integrate into the existing power system on the UAV. An IR LED was chosen for ease of separation from standard indoor light sources and for slightly better penetration of atmospheric disturbances, in this case smoke and dust in a damaged building or mine.

Figure 10 – 465 nm Blue LED

A pair of 465 nm blue LEDs were chosen for the forward LED indicators [5]. Initially a 625 nm LED, similar to the 850 nm, was chosen but it proved to be too close in spectrum to reliably identify, so the 465 nm blue LEDs were selected. The blue LEDs have a viewing angle of 45°, so two were mounted angled slightly away from each other to provide a wider detection angle. They have a forward voltage of 3.0 V at an 80 mA operating current, making them easy to integrate into the existing UAV power system.

In order to protect the Pixhawk autopilot of the UAV, which runs on a DC-DC switching 5 V regulator attached to the onboard 4S battery, an additional DC-DC switching 5 V regulator was installed to provide power solely for the tracking LEDs. This was done to ensure that any failure or shorting of the LEDs would not also lead to the failure of the Pixhawk autopilot.

The LEDs are hardwired to the 5 V output of the regulator with current limiting resistors in series on the input. When compared to the current drawn by the UAV in flight, the additional current from the LEDs does not amount to a large impact on flight or charging time.

Table 1 - UAV Power Budget

Average UAV Current in Flight (A)	Current Range in Flight (A)	Flight Time (Minutes)	Battery Capacity (Ah)
24	20-30	10-14	5.2
LED Current (A)	LED Current Consumption for flight (Ah)	% Battery Usage Per Flight	Flight Time Reduction (Minutes)
1.12	0.186 - 0.261	3.5-5.0%	0.25 - 0.5
			Charging Time Increase at 1C Charging Rate (Minutes)
			2.14 - 3.01

4.5. Clearpath Husky UGV

The landing/charging platform for the UAV, as well as the mounting location for the RaspberryPI and NoIR camera is a modified Husky UGV made by Clearpath Robotics [6]. The Husky is 99 cm x 67 cm x 39 cm and weighs 50 kg, with a maximum payload of 75 kg. It is powered by a 24 V 20 Ah Lead Acid battery, with 5 V, 12 V, and 24 V 5 A power supplies available. It has been modified with a landing platform for the UAV and an additional storage space under the landing platform for the charging system and supplemental charging batteries. The landing and charging platform and UGV modifications were designed and built by Sarah Hoffman for a previous Master's Project.

4.6. RFD900+ Serial Modem

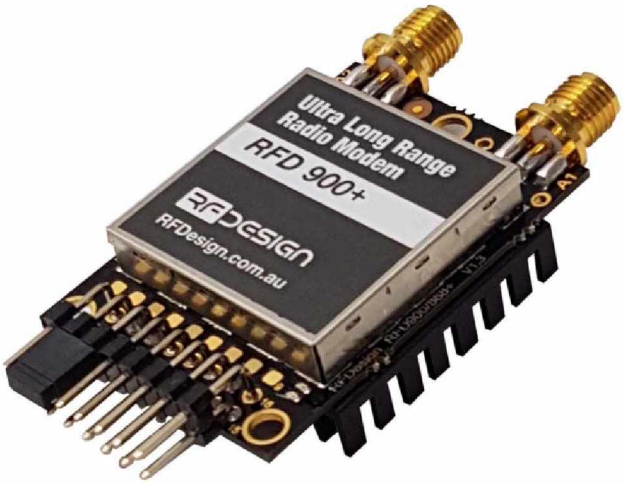

Figure 11 - RFD900+ Serial Modem

The RFD900+ serial modem was chosen to provide flight control communication between the RaspberryPI and the Pixhawk autopilot on the UAV. The RFD900+ is a 915MHz, 1W transmit power, serial radio [7]. This radio has been previously used in UAVs built for operations for ACUASI, UAF's UAV operations department, as well as numerous CEM drone class builds.

5. Method of Identification

5.1. OpenCV

This project uses OpenCV libraries to facilitate identification and tracking of the LEDs on the UAV. OpenCV, originally developed by Intel, is a free for use open source library released under a BSD license [8]. The fact that it is open source and very widely used in a variety of image tracking and object identification projects made it an ideal choice for this project, with many online tutorials and extensive documentation. The specific version used in this project is OpenCV2, built for Python 2.7.X. OpenCV also provides native support for the RaspberryPI and RaspberryPI NoIR camera.

5.2. Basic Program Operation

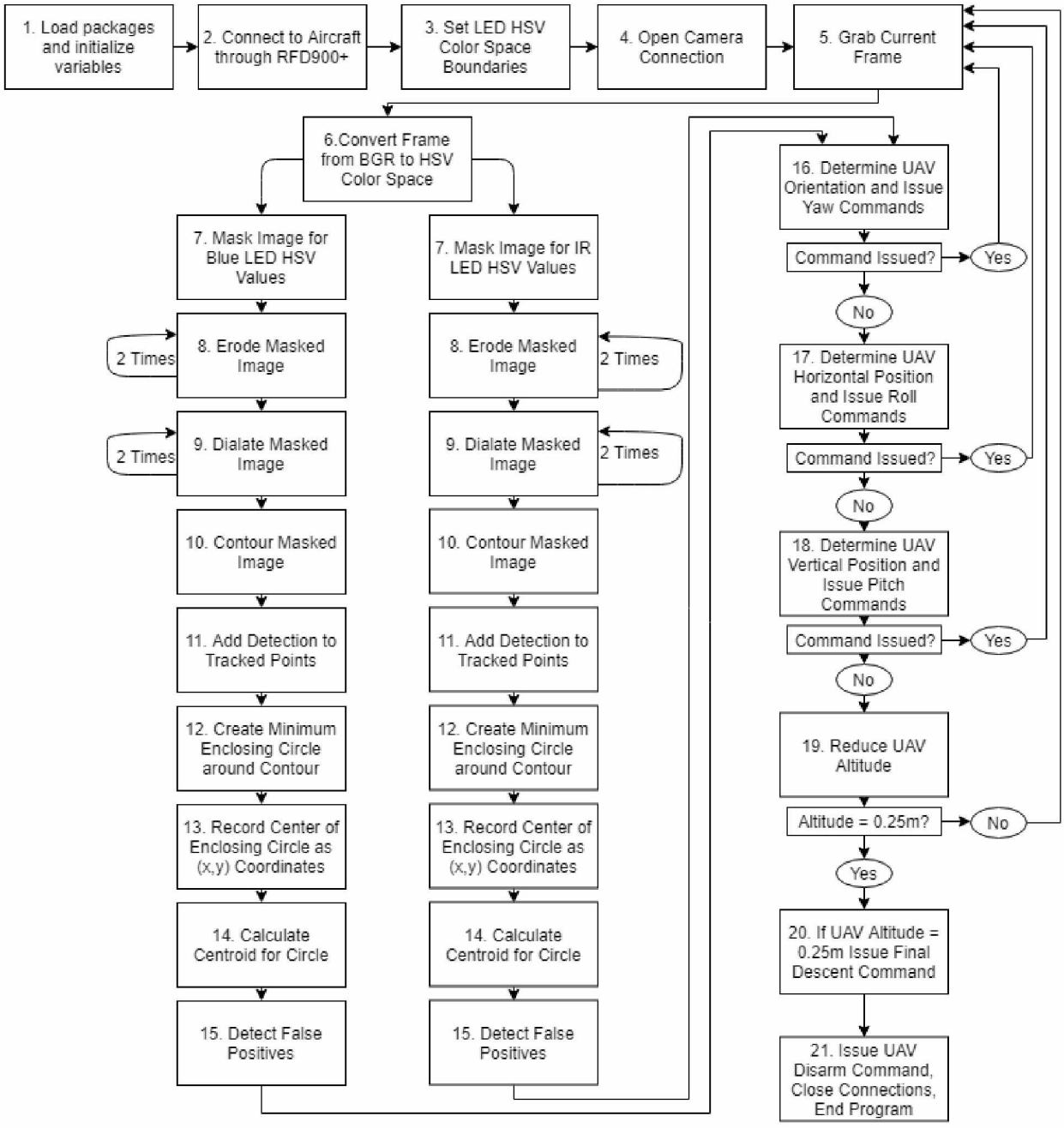

Figure 12 - Program Flowchart

The basic operation of the program is as follows:

1. The required OpenCV and Dronekit packages are loaded and variables are initialized.
2. Connection to UAV is opened through RFD900+.

3. The program sets an upper and lower boundary in HSV color space for the LEDs.
4. A connection to the camera is opened and a 2 second wait is observed to allow the camera sufficient time to start up and self adjust.
5. A loop is entered where the current frame is grabbed.
6. The grabbed frame is converted from BGR to HSV color space.
7. The image is masked with the cv2.inRange() function using the HSV color values for the LEDs.
8. Two passes of the function cv2.erode() are performed to reduce the noise in the mask as well as to draw in the edges of the detected LED.
9. Two passes of the function cv2.dialate() are performed on the mask to fill in the area of the detected LED.
10. The edges of the LED in the mask are defined using the function cv2.findContours().
11. If a positive detection has been made, then the detection will be added to the list of counted points for that LED.
12. A circular area enclosing the detected contour is created using the function cv2.minEnclosingCircle().
13. The center point of this circle is then calculated and saved as (x,y) coordinates.
14. The centroid of this circle is then calculated using the function cv2.moments() on the created minimum enclosing circle.
15. In order to reduce false positives, the detected circle must span at least 3 pixels, if it does the detected circle is then drawn on the output image to indicate detection.
16. If both the IR and blue LEDs have been detected, then the orientation of the UAV is determined using the relative position of the LEDs and commands to either rotate clockwise or counterclockwise are issued until the blue LED is towards the top of the frame and aligned with the IR LED. The loop is then restarted.

17. If no rotation commands have been issued this loop then the horizontal position relative to the center of the image is calculated based off the IR LED, with commands being issued to bring the UAV into the horizontal center of the image. The loop is then restarted.
18. If no rotation commands and no horizontal movement commands have been issued this loop then the vertical position relative to the center of the image is calculated, with commands being issued to the UAV to bring it into the center of the image. The loop is then restarted.
19. If no rotation, horizontal, or vertical movement commands have been issued this loop the UAV is commanded to reduce altitude. The loop is then restarted.
20. Once the UAV reports its altitude as less than 0.25 meters, the UAV is commanded to drastically cut motor RPM and descend. This aggressive control is necessary because of the large influence that ground effect has when so close to the landing platform.
21. The UAV is then considered to have landed, disarm commands are issued, and the program is finished.

5.3. LED HSV Color Space Determination

To start, the LEDs representation in HSV (Hue Saturation Value) color space was determined. This was done by creating a script with adjustable sliders that can be manipulated while the script is running. An unprocessed image is displayed as well as a masked image using the slider's HSV color values.

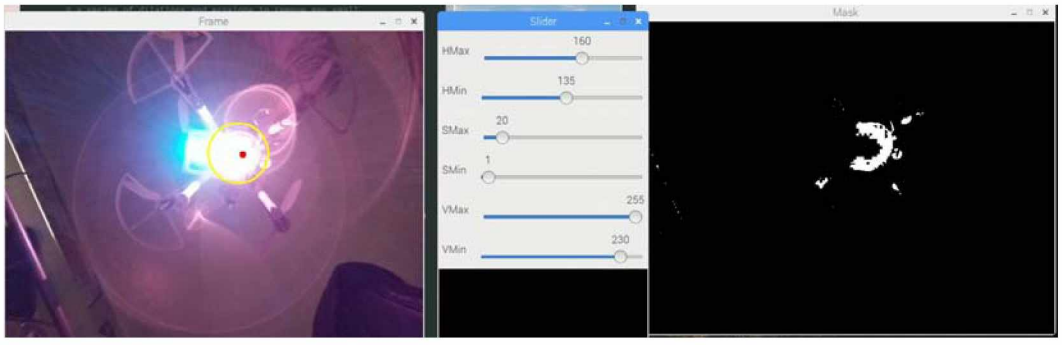

Figure 13 - HSV Masking Slider Output

The minimum and maximum values for the HSV mask are set while viewing the output masking to see how well the desired LED is selected. When the best values are found the program is ended, with the final minimum and maximum values printed to the terminal for future use.

Figure 14 - HSV Masking Slider Final Values

This procedure is then repeated for the blue tracking LEDs.

5.4. LED Tracking and Position Determination

To begin, the program loads the CV2 packages that will be used for this program.

```
from collections import deque
from imutils.video import VideoStream
import numpy as np
import argparse
import cv2
import imutils
import time
```

Figure 15 - CV2 Packages Used for LED Identification and Tracking

Variables are initialized for position information for each LED. The HSV color space values, determined using the HSV Masking Slider script, are then set.

```
BlueLower = (69, 10, 230)
BlueUpper = (91, 91, 255)
IRLower = (135, 1, 230)
IRUpper = (160, 20, 255)
```

Figure 16 - HSV Color Space Values for Tracking LEDs

The NoIR camera is then initialized and given 2 seconds to warm up and stabilize.

```
# grab the reference
# to the webcam

        vs = VideoStream(src=0).start()

# allow the camera to warm up
time.sleep(2.0)
```

Figure 17 - Starting the NoIR Camera

With the camera ready, the frame grab loop is entered. This loop will begin with grabbing the most recent frame from the camera and then converting it from BGR to HSV color space using the cv2.cvtColor() function.

```
# grab the current frame
frame = vs.read()
#convert the frame to HSV color space
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

Figure 18 - Grabbing the Current Frame and Converting to HSV color

Now that a frame is ready and converted to HSV color, the HSV masking values for each LED can be separately applied and each mask can be eroded and dilated to remove noise and fill in gaps in the mask. This step required the most fine tuning, as different methods and number of passes were tested to balance performance versus clarity of the masked LED. It was important to keep the computation time low enough that it could be performed multiple times each second to provide smooth updates for the position of the tracked LEDs. The display of the mask is only for testing purposes and is not necessary during normal operations.

```
maskBlue = cv2.inRange(hsv, BlueLower, BlueUpper)
maskIR = cv2.inRange(hsv, IRLower, IRUpper)
cv2.imshow("MaskBlue", maskBlue)
cv2.imshow("MaskIR", maskIR)
maskIR = cv2.erode(maskIR, None, iterations=2)
maskBlue = cv2.erode(maskBlue, None, iterations=2)
maskBlue = cv2.dilate(maskBlue, None, iterations=2)
maskIR = cv2.dilate(maskIR, None, iterations=2)
```

Figure 19 - Masking the Grabbed Frame for each LEDs HSV color values

With the masking of the individual frames for each LED complete, contours are calculated for each frame to surround the remaining "blobs" in the image.

```
cntsBlue = cv2.findContours(maskBlue.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
cntsBlue = cntsBlue[0] if imutils.is_cv2() else cntsBlue[1]

cntsIR = cv2.findContours(maskIR.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
cntsIR = cntsIR[0] if imutils.is_cv2() else cntsIR[1]
```

Figure 20 - Contour Calculations

Figure 21 - LED Frames Final Mask

Using the calculated contours, a minimum enclosing circle can be calculated for each frame, from which the center of the "blob" can be extracted. This minimum enclosing circle is also used to draw the circles seen on the output frame during testing in order to visualize how well the LED is being tracked.

```
((x, y), radius) = cv2.minEnclosingCircle(cBlue)

MBlue = cv2.moments(cBlue)
centerBlue = (int(MBlue["m10"] / MBlue["m00"]), int(MBlue["m01"] / MBlue["m00"]))
```

Figure 22 - Calculating the Minimum Enclosing Circle

Now that the (x,y) position for each LED is calculated, the orientation and position of the UAV can be determined and used to command the UAV to move in the required direction.

6. UAV Control Process

Control of the UAV is achieved by using the Dronekit-Python API [9]. Dronekit is an open source project for receiving telemetry information and for sending control commands to an ArduPilot based flight controller, such as the Pixhawk, using MAVlink messages. It is available for use under the Apache v2 open source license. Since the UAV will not have GPS signal, making standard relative direction commands

impossible, Dronekit will be used to directly send commands that override PWM inputs to directly control the UAV, much like the signals received from a hand controller used by a human pilot. Because of the inherent instability of controlling a UAV in this manner, multiple small inputs will be used to slowly "nudge" the UAV in the desired orientation and direction.

6.1. Dronekit-Python

To start, the necessary Dronekit packages are loaded in the program.

```
import serial
import dronekit
from dronekit import connect
```

Figure 23 - Loading Dronekit Package

Next, an open serial port is created to the RFD900+ and then the UAV is connected to, waiting until the connection is confirmed.

```
port = serial.Serial("dev/ttyS0", baudrate=57600, timeout=3.0)
vehicle = connect(port, wait_ready=True)
```

Figure 24 - Connecting to the UAV

Once the tracking LEDs positions are determined, the presence of tracked points is confirmed. If tracking is confirmed, the flight mode of the UAV is changed to Altitude Hold and the mode change is flagged so it doesn't attempt every loop. In this mode the UAV will maintain constant altitude as measured by its onboard laser rangefinder. In this mode, RC commands are interpreted as rate commands; for example, raising the throttle will command the UAV to ascend at a rate proportional to the level of the input.

```
if len(cntsIR) > 0 and len(cntsBlue) > 0:
        #Now that the UAV has been aquired, change mode to Altitude hold if
        #not already done
        if mode == 0:
                global mode
                vehicle.mode = VehicleMode("ALTHOLD")
                mode = 1
```

Figure 25 - Setting UAV Flightmode to AltHold

Now that the flight mode is properly set, the rotational orientation of UAV is determined. Commands are issued to rotate the UAV until the Blue LED is towards the top of the frame and the two tracked LEDs are within vertical alignment. With the Blue LED towards the top of the screen, the direction the UAV will travel when given movement commands is known. During each loop only one type of command will be issued. This is to prevent commands for rotate and directional travel from being sent at the same time.

```
if int(x) > (int(xIR) + 10) and int(y) < int(yIR):
        vehicle.channels.overrides['4']=1525
        #Display commands on screen for testing
        #font = cv2.FONT_HERSHEY_SIMPLEX
        #cv2.putText(frame,'CCW',(10,100), font, 2, (0,0,255),2,cv2.LINE_AA)
        global rotate
        rotate = 1

if int(x) < (int(xIR) - 10) and int(y) < int(yIR):
        vehicle.channels.overrides['4']=1475
        #Display commands on screen for testing
        #font = cv2.FONT_HERSHEY_SIMPLEX
        #cv2.putText(frame,'CW',(400,100), font, 2, (0,0,255),2,cv2.LINE_AA)

        global rotate
        rotate = 1
if int(y) < int(yIR):
        #if the LEDs are aligned but the Blue LED is not on top, rotate the craft

        vehicle.channels.overrides['4']=1475
        global rotate
        rotate = 1
```

Figure 26 - Determining and Commanding UAV Rotation

If no rotational commands have been issued this loop, commands are sent to bring the UAV into the center of the frame with respect to its position in the y-plane of the frame.

```
if rotate == 0:
        if int(yIR) > 300:
                vehicle.channels.overrides['1']=1525
                #Display commands on screen for testing
                #font = cv2.FONT_HERSHEY_SIMPLEX
                #cv2.putText(frame,'BACK',(200,400), font, 2, (0,0,255),2,cv2.LINE_AA)
                global horiz
                horiz = 1

        if int(yIR) < 250:
                vehicle.channels.overrides['1']=1475
                #Display commands on screen for testing
                #font = cv2.FONT_HERSHEY_SIMPLEX
                #cv2.putText(frame,'FORWARD',(200,100), font, 2, (0,0,255),2,cv2.LINE_AA)
                global horiz
                horiz = 1
```

Figure 27 - Determining and Commanding UAV Horizontal Position

If no rotational or horizontal movement commands have been issued this loop, commands are sent to bring the UAV into the center of the frame with respect to its position on the x-plane of the frame.

```
if horiz == 0:
        if int(xIR) > 400:
                vehicle.channels.overrides['2']=1525
                #Display commands on screen for testing
                #font = cv2.FONT_HERSHEY_SIMPLEX
                #cv2.putText(frame,'LEFT',(10,200), font, 2, (0,0,255),2,cv2.LINE_AA)
                global vert
                vert = 1

        if int(xIR) < 350:
                vehicle.channels.overrides['2']=1475
                #Display commands on screen for testing
                #font = cv2.FONT_HERSHEY_SIMPLEX
                #cv2.putText(frame,'RIGHT',(400,200), font, 2, (0,0,255),2,cv2.LINE_AA)
                global vert
                vert = 1
```

Figure 28 - Determining and Commanding UAV Vertical Position

Once the UAV has completed a loop with no movement commands being needed, the UAV is considered to be directly above the landing platform and is commanded to descend. It was discovered during the testing phase that in the final phase of the landing, when the UAV is approximately 0.25 m above the landing platform, that ground effects have a large detrimental impact on the UAV's ability to maintain stable flight. The solution to this was to command a sharp descent when this range above the landing platform is reached. Once the final command to descend has been issued, the UAV is disarmed, the connection closed, and the loop is exited.

```
if vehicle.rangefinder.distance < 0.25:

        #if the distance to the platform is less than 0.25 meter, command the aircraft
        #to quickly descend.  This is to overcome the ground effect caused by the
        #landing platform
        vehicle.mode = VehicleMode("LAND")
        #wait for the aircraft to land
        time.sleep(1.0)
        #disarm the aircraft and close the connection
        vehicle.armed = False
        vehicle.close()
        #stop the loop
        break


else
        #if no control commands have been issued this loop, command the aircraft to descend
        vehicle.channels.overrides['3']=1475
      #Display commands on screen for testing
      #font = cv2.FONT_HERSHEY_SIMPLEX
      #cv2.putText(frame,'DOWN',(400,200), font, 2, (0,0,255),2,cv2.LINE_AA)
```

Figure 29 - Commanding the UAV to Descend when Centered Above Landing Platform

7. Future Improvements

7.1. Landing Platform and UAV Improvements

One of the major issues discovered during the testing phase of the project was the flight instability caused in the final descent phase by ground effects between the UAV and the large, flat landing platform. This is especially problematic because the UAVs landing legs descend into the platform by several centimeters. One solution to this problem would be to break up the flat plane of the landing platform, either with large gaps to allow air flow, or to redesign the conical receptacles into a design like inverted pyramids for each landing leg, with longer more gradual slopes to help better guide the landing legs into place.

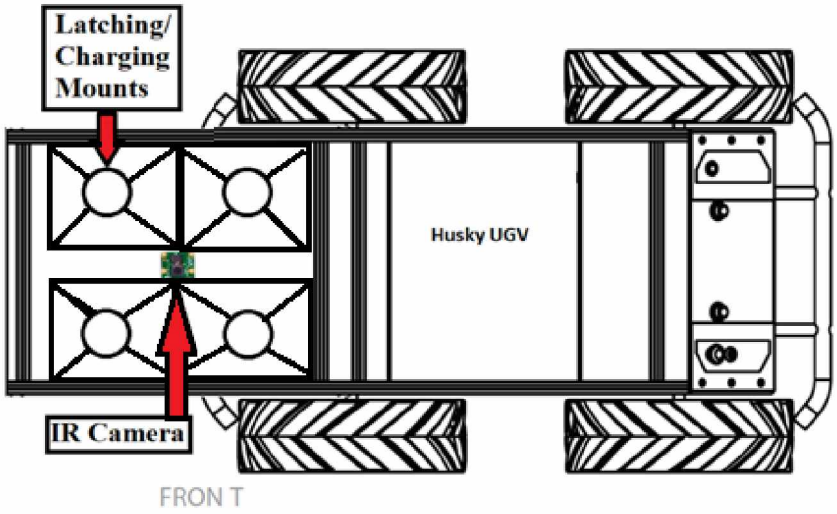

Figure 30 - Modified Landing Platform with More Gradual Slopes

7.2. LED Bloom Mitigation

The 1 W, 850 nm IR LED, while easy to detect at a distance, caused excessive saturation and blooming of the camera image during the final landing phase when the UAV got close to the landing platform. This could be mitigated by current limiting, and therefore reducing the brightness, of the LED during the landing phase using a PWM controlled MOSFET. This PWM signal could easily be sourced from the Pixhawk autopilot, as it has several unused PWM outputs, and be tied directly to the onboard altitude measurements. The RFD900+ also has a PWM output that could be used, which would be controlled by the RaspberryPI.

7.3. PID Control

Utilizing a PID control loop for horizontal and vertical position would increase stability and decrease overall landing time. It would make variable rate commands easier, meaning more aggressive commands the more distance needed to travel. PID control would also provide better resilience to external forces acting upon the UAV, such as air currents inside the mine/building.

8. Appendices

8.1. Acronyms

UAV – Unmanned Aerial Vehicle

GPS – Global Positioning System

IR – Infra Red

LED – Light Emitting Diode

UGV – Unmanned Ground Vehicle

NoIR – No Infra Red cut filter

UART – Universal Asynchronous Receiver Transmitter

nm – nanometer

Ah – Ampere Hour

Wh – Watt Hour

LiPo – Lithium Polymer

BSD License – Berkely Software Distribution License

BGR – Blue Green Red color space

HSV – Hue Saturation Value color space

MAVLink – Micro Air Vehicle Link

RC – Radio Controlled

UAF – University of Alaska Fairbanks

PWM – Pulse Width Modulation

MOSFET – Metal-Oxide Semiconductor Field-Effect Transistor

PID – Proportional Integral Derivative

8.2. References

[1] "Raspberry Pi 3 Model B ," *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Accessed: 19-Apr-2019].

[2] "Pi NoIR Camera V2," *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org/products/pi-noir-camera-v2/. [Accessed: 19-Apr-2019].

[3] "Pixhawk 1," *Pixhawk 1 · PX4 v1.8.2 User Guide*. [Online]. Available: https://docs.px4.io/en/flight_controller/pixhawk.html. [Accessed: 19-Apr-2019].

[4] *IR-1W-850 Datasheet*. [Online]. Available: https://d114hh0cykhyb0.cloudfront.net/pdfs/IR-1W-850.pdf. [Accessed: 19-Apr-2019].

[5] *YSL-R1042B5C-D13 Blue LED Datasheet*. [Online]. Available: https://www.sparkfun.com/datasheets/Components/LED/Blue-10mm.pdf. [Accessed: 19-Apr-2019].

[6] "Husky UGV - Outdoor Field Research Robot by Clearpath," *Clearpath Robotics*. [Online]. Available: https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/. [Accessed: 19-Apr-2019].

[7] "RFD 900 Modem," *RFDesign*. [Online]. Available: http://store.rfdesign.com.au/rfd-900p-modem/. [Accessed: 19-Apr-2019].

[8] "OpenCV," *OpenCV*, 17-Apr-2019. [Online]. Available: https://opencv.org/. [Accessed: 19-Apr-2019].

[9] "Dronekit for Python," *Dronekit*, 15-Feb-2019. [Online]. Available: https://python.dronekit.io/. [Accessed: 19-Apr-2019].

8.3. Code

```
 # import the packages
from collections import deque
from imutils.video import VideoStream
import numpy as np
import argparse
import cv2
import imutils
import time


import serial
import dronekit
from dronekit import connect



global x
global xIR
global y
global yIR
global rotate
global horiz
global vert
global mode

x = 0
xIR = 0
y = 0
yIR = 0
rotate = 0
mode = 0
vert = 0

port = serial.Serial("dev/ttyS0", baudrate=57600, timeout=3.0)
vehicle = connect(port, wait_ready=True)
vehicle.wait_ready('autopilot_version')


# define the lower and upper boundaries of the LEDs
# in the HSV color space, then initialize the
# list of tracked points
BlueLower = (69, 10, 230)
BlueUpper = (91, 91, 255)
IRLower = (135, 1, 230)
IRUpper = (160, 20, 255)

ptsBlue = deque(maxlen=50)
ptsIR = deque(maxlen=50)

# grab the reference
```

```
# to the webcam

vs = VideoStream(src=0).start()

# allow the camera to warm up
time.sleep(2.0)

# keep looping
while True:
        #clear all RC override commands
        vehicle.channels.overrides = {}

        #initialize command variables
        global vert
        global rotate
        global horiz
        horiz = 0
        rotate = 0
        vert = 0
        # grab the current frame
        frame = vs.read()

        # Convert it to the HSV
        # color space

        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # construct a mask for the color, then perform
        # a series of dilations and erosions to remove any small
        # blobs left in the mask
        maskBlue = cv2.inRange(hsv, BlueLower, BlueUpper)
        maskIR = cv2.inRange(hsv, IRLower, IRUpper)

        maskIR = cv2.erode(maskIR, None, iterations=2)
        maskBlue = cv2.erode(maskBlue, None, iterations=2)
        maskBlue = cv2.dilate(maskBlue, None, iterations=2)
        maskIR = cv2.dilate(maskIR, None, iterations=2)
        cv2.imshow("MaskBlue", maskBlue)
        cv2.imshow("MaskIR", maskIR)
        # find contours in the mask and initialize the current
        # (x, y) center of the "blob"
        cntsBlue = cv2.findContours(maskBlue.copy(), cv2.RETR_EXTERNAL,
                cv2.CHAIN_APPROX_SIMPLE)
        cntsBlue = cntsBlue[0] if imutils.is_cv2() else cntsBlue[1]

        cntsIR = cv2.findContours(maskIR.copy(), cv2.RETR_EXTERNAL,
                cv2.CHAIN_APPROX_SIMPLE)
        cntsIR = cntsIR[0] if imutils.is_cv2() else cntsIR[1]

        centerBlue = None
        centerIR = None



        # only proceed if at least one contour was found
        if len(cntsBlue) > 0:
                # find the largest contour in the mask, then use
```

```
        # it to compute the minimum enclosing circle and
        # centroid
        cBlue = max(cntsBlue, key=cv2.contourArea)

        global x
        global y
        ((x, y), radius) = cv2.minEnclosingCircle(cBlue)


        MBlue = cv2.moments(cBlue)
          centerBlue = (int(MBlue["m10"] / MBlue["m00"]),
          int(MBlue["m01"] / MBlue["m00"]))

        if radius > 3:
                # draw the circle and centroid on the frame,
                 # then update the list of tracked points
                cv2.circle(frame, (int(x), int(y)), int(radius),
                         (0, 255, 255), 2)
                cv2.circle(frame, centerBlue, 5, (0, 0, 255), -1)



if len(cntsIR) > 0:
        cIR = max(cntsIR, key=cv2.contourArea)
        global xIR
        global yIR
        ((xIR,yIR), radius2) = cv2.minEnclosingCircle(cIR)
        #print("IR")
        #print(int(xIR),int(yIR))
        MIR = cv2.moments(cIR)
          centerIR = (int(MIR["m10"] / MIR["m00"]), int(MIR["m01"] /
          MIR["m00"]))

        # only proceed if the radius meets a minimum size
        if radius2 > 3:
                 # draw the circle and centroid on the frame,
                 # then update the list of tracked points
                cv2.circle(frame, (int(xIR),   int(yIR)),
                int(radius2),(0, 255, 255), 2)
                 cv2.circle(frame, centerIR, 5, (0, 0, 255), -1)



# update the points queue
ptsBlue.appendleft(centerBlue)
ptsIR.appendleft(centerIR)

# ------------------------------------------------------------------
    # THIS SECTION IS FOR TESTING ONLY --- IT ADDS A PATH ON SCREEN
    FOR TRACKED POINTS
# loop over the set of tracked points
# for i in range(1, len(ptsBlue)):
        # if either of the tracked points are None, ignore
        # them
        #if ptsBlue[i - 1] is None or ptsBlue[i] is None:
        #        continue
```

```
        # otherwise, compute the thickness of the line and
        # draw the connecting lines
          #thickness = int(np.sqrt(args["buffer"] / float(i + 1)) *
          2.5)
        #cv2.line(frame, ptsBlue[i - 1], ptsBlue[i], (0, 0, 255),
        thickness)


#for i in range(1, len(ptsIR)):
        # if either of the tracked points are None, ignore
        # them
        #if ptsIR[i - 1] is None or ptsIR[i] is None:
        #        continue

        # otherwise, compute the thickness of the line and
        # draw the connecting lines
        #thickness = int(np.sqrt(args["buffer"] / float(i + 1)) *
         2.5)
        #cv2.line(frame, ptsIR[i - 1], ptsIR[i], (0, 0, 255),
         thickness)
#-------------------------------------------------------------------
         ----
        #Now that the UAV has been aquired, change mode to Altitude
         hold if
        #not already done
if mode == 0:
        global mode
        vehicle.mode = VehicleMode("ALTHOLD")
        mode = 1


if len(cntsIR) > 0 and len(cntsBlue) > 0:
        if int(x) > (int(xIR) + 10) and int(y) < int(yIR):
                vehicle.channels.overrides['4']=1525
                #Display commands on screen for testing
                #font = cv2.FONT_HERSHEY_SIMPLEX
                #cv2.putText(frame,'CCW',(10,100), font, 2,
                 (0,0,255),2,cv2.LINE_AA)
                global rotate
                rotate = 1

        if int(x) < (int(xIR) - 10) and int(y) < int(yIR):
                vehicle.channels.overrides['4']=1475
                #Display commands on screen for testing
                #font = cv2.FONT_HERSHEY_SIMPLEX
                #cv2.putText(frame,'CW',(400,100), font, 2,
                (0,0,255),2,cv2.LINE_AA)

                global rotate
                rotate = 1
        if int(y) > int(yIR):
                #if the LEDs are aligned but the Blue LED is not on
                 top, rotate the craft

                vehicle.channels.overrides['4']=1475
                global rotate
                rotate = 1
```

```
if rotate == 0:
        if int(yIR) > 300:
                vehicle.channels.overrides['1']=1475
                #Display commands on screen for testing
                #font = cv2.FONT_HERSHEY_SIMPLEX
                #cv2.putText(frame,'BACK',(200,400), font, 2,
                (0,0,255),2,cv2.LINE_AA)
                global horiz
                horiz = 1

        if int(yIR) < 250:
                vehicle.channels.overrides['1']=1475
                #Display commands on screen for testing
                #font = cv2.FONT_HERSHEY_SIMPLEX
                #cv2.putText(frame,'FORWARD',(200,100), font,
                2, (0,0,255),2,cv2.LINE_AA)
                global horiz
                horiz = 1

        if horiz == 0:
                if int(xIR) > 400:
                        vehicle.channels.overrides['2']=1475
                        #Display commands on screen for
                        testing
                        #font = cv2.FONT_HERSHEY_SIMPLEX
                        #cv2.putText(frame,'LEFT',(10,200),
                         font, 2, (0,0,255),2,cv2.LINE_AA)
                        global vert
                        vert = 1

                if int(xIR) < 350:
                        vehicle.channels.overrides['2']=1475
                        #Display commands on screen for
                         testing
                        #font = cv2.FONT_HERSHEY_SIMPLEX
                        #cv2.putText(frame,'RIGHT',(400,200),
                        font, 2, (0,0,255),2,cv2.LINE_AA)
                        global vert
                        vert = 1

                if vert == 0:
                         if vehicle.rangefinder.distance <=
                         0.25:

                                #if the distance to the
                                platform is less than 0.25
                                meter, command the aircraft
                                #to quickly descend.  This
                                is to overcome the ground
                                effect caused by the
                                #landing platform

                                 vehicle.mode =
                                 VehicleMode("LAND")
                                #wait for the aircraft to
                                 land
                                time.sleep(1.0)
```

```
                                        #disarm the aircraft and
                                        close the connection
                                        vehicle.armed = False
                                        vehicle.close()
                                        #stop the loop
                                        break


                                else
                                        #if no control commands have
                                        been issued this loop,
                                        command the aircraft to
                                        descend

                                        vehicle.channels.overrides['3
                                        ']=1475
                                        #Display commands on screen
                                         for testing
                                        #font =
                                        cv2.FONT_HERSHEY_SIMPLEX

                                        #cv2.putText(frame,'DOWN',(40
                                        0,200), font, 2,
                                        (0,0,255),2,cv2.LINE_AA)


        # show the frame to our screen
        #cv2.imshow("Frame", frame)
        #key = cv2.waitKey(1) & 0xFF

        # if the 'q' key is pressed, stop the loop
        #if key == ord("q"):
        #       break


# release the camera
vs.release()

# close all windows
cv2.destroyAllWindows()
```